Linguagem C++:
Variáveis e expressões

Linguagens de programação

- Linguagem de Máquina
  - Computador entende apenas pulsos elétricos
  - Presença ou não de pulso
  - 1 ou 0
- Tudo no computador deve ser descrito em termos de 1's ou 0's (binário)
  - Difícil para humanos ler ou escrever
  - 00011110 = 30

Linguagens de programação

- Linguagem *Assembly*
  - Uso de mnemônicos
  - Conjunto de 0's e 1's é agora representado por um código
  - 10011011 -> ADD

- Linguagem *Assembly* - Problemas
  - Requer programação especial (*Assembly*)
  - Conjunto de instruções varia com o computador (processador)
  - Ainda é muito difícil programar

Linguagens de programação

- Linguagens de Alto Nível
  - Programas são escritos utilizando uma linguagem parecida com a linguagem humana
  - Independente da arquitetura do computador
  - Mais fácil programar
  - Uso de compiladores

Linguagens de programação

- Primórdios
  - Uso da computação para cálculos de fórmulas
  - Fórmulas eram traduzidas para linguagem de máquinas
  - Por que não escrever programas parecidos com as fórmulas que se deseja computar?

Linguagens de programação

- FORTRAN (FORmula TRANsform)
  - Em 1950, um grupo de programadores da IBM liderados por John Backus produz a versão inicial da linguagem;
  - Primeira linguagem de alto nível;
- Várias outras linguagens de alto nível foram criadas
  - Algol-60, Cobol, Pascal, etc

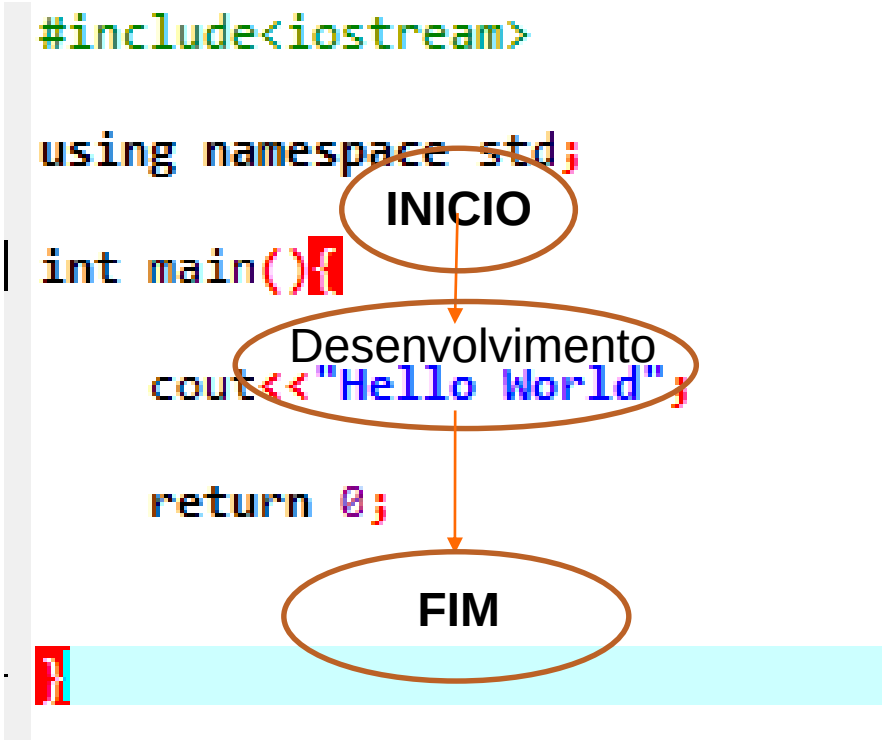
Linguagem C++

- Uma das mais bem sucedidas foi uma linguagem chamada C++
  - Criada em 1980 nos laboratórios por Bjarne Stroustup
  - Derivada da linguagem C, com intuito de ser orientada ao objeto.

Primeiro programa em C++

```
#include<iostream>

using namespace std;

int main(){

    cout<<"Hello World";

    return 0;


}
```

Primeiro programa em C++



Primeiro programa em C++

```
#include<iostream>

using namespace std;

int main(){

    cout<<"Hello World";

    return 0;


}
```

#include é utilizado para adicionar uma biblioteca ao seu programa

Um namespace é uma região declarativa que fornece um escopo para os identificadores (os nomes de tipos, funções, variáveis, etc) dentro dela.

Int main() é a função principal do seu programa

Cout é a função de console output, ou seja, saída.

Return é o valor de retorno da função main, nesse caso 0.

Primeiro programa em C+

- Por que escrevemos programas?
  - Temos dados ou informações que precisam ser processados;
  - Esse processamento pode ser algum cálculo ou pesquisa sobre os dados de entrada;
  - Desse processamento, esperamos obter alguns resultados (Saídas);

COMENTÁRIOS

- Permitem adicionar uma descrição sobre o programa. São ignorados pelo compilador.

```
#include<iostream>

using namespace std;
// Esta é a função principal do meu programa
int main(){
/* utilizamos o // para comentar em apenas uma linha e o /* para iniciar uma
sequencia de linahs comentadas encerrando o mesmo com */

    cout << "Hello World";

    return 0;

    }
```

Variáveis

- Matemática
  - é uma entidade capaz de representar um valor ou expressão;
  - pode representar um número ou um conjunto de números
  - $f(x) = x^2$

Variáveis

- Computação
  - Posição de memória que armazena uma informação
  - Pode ser modificada pelo programa
  - Deve ser **definida** antes de ser usada

Declaração de variáveis

- Precisamos informar ao programa quais dados queremos armazenar
- Precisamos também informar o que são esses dados (qual o tipo de dado)
  - Um nome de uma pessoa
    - Uma cadeia de caracteres (“André” - 5 caracteres)
  - O valor da temperatura atual
    - Um valor numérico (com casas decimais)
  - A quantidade de alunos em uma sala de aula
    - Um valor numérico (número inteiro positivo ou zero)
  - Se um assento de uma aeronave está ocupado
    - Um valor lógico (ocupado: verdadeiro / desocupado: falso)

Variáveis

- Declaração de variáveis em C
  - Tipo da variável nome da variável
  - Ex: int a;
- Propriedades
  - Nome
    - Pode ter um ou mais caracteres
    - Nem tudo pode ser usado como nome
  - Tipo
    - Conjunto de valores aceitos

Variáveis

- Nome
  - Deve iniciar com letras ou underscore (_);
  - Caracteres devem ser letras, números ou underscores;
  - Palavras chave não podem ser usadas como nomes;
  - Letras maiúsculas e minúsculas são consideradas diferentes

VARIÁVEIS

- Nome
  - Não utilizar espaços nos nomes
    - Exemplo: nome do aluno, temperatura do sensor,
  - Não utilizar acentos ou símbolos
    - Exemplos: garça, tripé, o,Θ
  - Não inicializar o nome da variável com números
    - Exemplos: 1A, 52, 5ª
  - Underscore pode ser usado
    - Exemplo: nome_do_aluno : caracter
  - Não pode haver duas variáveis com o mesmo nome*

VARIÁVEIS

- Lista de palavras chave

auto	break	case	char	const	continue	do	double
else	for	int	union	static	default	void	return
enum	goto	long	unsigned	struct	extern	while	sizeof
float	if	short	volatile	switch	register	typeof	

Variáveis

- Quais nomes de variáveis estão corretos?
  - Contador
  - contador1
  - comp!
  - .var
  - Teste_123
  - _teste
  - int
  - int1
  - 1contador
  - -x
  - Teste-123
  - x&

Variáveis

- Corretos:
  - Contador, contador1,Teste_123, _teste, int1
- Errados
  - comp!, .var, int, 1contador, -x, Teste-123, x&

Variáveis

- Tipo
  - Define os valores que ela pode assumir e as operações que podem ser realizadas com ela
- Exemplo
  - tipo **int** recebe apenas valores inteiros
  - tipo **float** armazena apenas valores reais

TIPOS BÁSICOS EM C++

- **char**: um byte que armazena o código de um caractere do conjunto de caracteres local
  - ***caracteres sempre ficam entre 'aspas simples'!***

```
char sexo; // pode receber'M' ou 'F'
char UnidadeTemperatura; //pode receber'C' para Celsius
                         //ou 'F' para Fahrenheit
char opcoes; // pode ser '1', '2' ,'3' ou '4'
```

- **int**: um inteiro cujo tamanho depende do processador, tipicamente 32 ou 64 bits

```
int NumeroAlunos;
int Idade;
int NumeroContaCorrente;
int N = 10; // o variável N recebe o valor 10
```

Tipos básicos em C++

- Números reais
  - Tipos: *float*, *double* e *long double*
  - A parte decimal usa **ponto** e **não vírgula**!
  - **float**: um número real com precisão simples

```
float Temperatura; // por exemplo, 23.30
float MediaNotas; // por exemplo, 7.98
float TempoTotal; // por exemplo, 0.0000000032 (s)
```

  - **double**: um número real com precisão dupla
    - Números muito grandes ou muito pequenos

```
double DistanciaGalaxias; // número muito grande
double MassaMolecular; // em Kg, número muito pequeno
double BalancoEmpresa; // valores financeiros
```

Tipos básicos em C++

- Números reais
  - Pode-se escrever números reais usando notação científica

```
double TempoTotal = 0.000000003295;

// notação científica
double TempoTotal = 3.2950e-009;
```

equivale à $3{,}295 \times 10^{-9}$

VARIÁVEIS

Tipo	Bits	Intervalo de valores
char	8	-128 A 127
unsigned char	8	0 A 255
signed char	8	-128 A 127
int	32	-2.147.483.648 A 2.147.483.647
unsigned int	32	0 A 4.294.967.295
signed int	32	-32.768 A 32.767
short int	16	-32.768 A 32.767
unsigned short int	16	0 A 65.535
signed short int	16	-32.768 A 32.767
long int	32	-2.147.483.648 A 2.147.483.647
unsigned long int	32	0 A 4.294.967.295
signed long int	32	-2.147.483.648 A 2.147.483.647
float	32	1,175494E-038 A 3,402823E+038
double	64	2,225074E-308 A 1,797693E+308
long double	96	3,4E-4932 A 3,4E+4932

ATRIBUIÇÃO

- Operador de Atribuição: =
  - nome_da_variável = expressão, valor ou constante;

O operador de atribuição "=" armazena o valor ou resultado de uma expressão contida à sua **direita** na variável especificada à sua **esquerda**.

- Ex.:

```
int main( ){
    int x = 5; // x recebe 5
    int y;
    y = x + 3; // y recebe x mais 3

    return 0;
}
```

  - A linguagem C++ suporta múltiplas atribuições
    - x = y = z = 0;

COMANDO DE SAÍDA

- **Cout**
  - Comando que realiza a exibição, do valor atribuído a uma variável ou a uma função, no console principal, nesse caso Tela.

```
#include<iostream>

using namespace std;

int main(){
    int a= 3;
    int b= 5;
    cout<< a << endl << b << endl << a+b;

    return 0;

}
```

Comando de entrada

- **cin**
  - Comando que realiza a leitura dos dados da entrada padrão (no caso o teclado)

```
#include<iostream>

using namespace std;

int main(){
    int a;
    int b;
    cin >> a >> b;
    cout<< a << endl << b << endl << a+b;

    return 0;

}
```

Exercícios

- 1 – Escreva um programa que faça a soma entre os números inteiros 10 e -20 inicializados em variáveis e mostre o resultado em tela.
- 2 – Escreva um programa que leia 2 números inteiros que faça a soma ou a subtração entre eles.

Constantes

- Como uma variável, uma constante também armazena um valor na memória do computador.
- Entretanto, esse valor não pode ser alterado: é constante.
- Para constantes é obrigatória a atribuição do valor.

CONSTANTES

- Usando **#define**
  - Você deverá incluir a diretiva de pré-processador **#define** antes de início do código:
  - **Cuidado: não colocar “;”**

```
#define PI 3.1415
```

- Usando **const**
  - Usando **const**, a declaração não precisa estar no início do código
  - A declaração é igual a de uma variável inicializada

```
const double pi = 3.1415;
```

Sequências de escape

- São constantes predefinidas
- Elas permitem o envio de caracteres de controle não gráficos para dispositivos de saída
- Outro escape é o endl, o qual deve ser digitado após a sequência de output
  - Cout << a << endl;

Código	Comando
\a	som de alerta (bip)
\b	retrocesso (backspace)
\n	nova linha (new line)
\r	retorno de carro (carriage **return**)
\v	tabulação vertical
\t	tabulação horizontal
\'	apóstrofe
\"	aspa
\\	barra invertida (backslash)
\f	alimentação de folha (form feed)
\?	símbolo de interrogação
\0	caractere nulo (cancela a escrita do restante)

Sequências de escape

- Exemplo

```
#include<iostream>

using namespace std;

int main(){
    cout << "Hello World\n";
    cout << "Hello \n World\n";
    cout << "Hello \\ World\n";
    cout << "\" Hello World\"\n";

    return 0;

    }
```

- Saída

```
Hello World
Hello
World
Hello \ World
"Hello World"
```

Tipos Booleanos em C++

- Um tipo booleano pode assumir dois valores:
  - Verdadeiro ou falso(true ou false)

- Exemplos:

  - Bool a= true;
  - Bool b= 0; (O 0 significa falso e o 1 verdadeiro)

Operadores

- Os operadores são usados para desenvolver diferentes tipos de operações. Com eles podemos:
  - Realizar operações matemáticas com suas variáveis.
  - Realizar operações de comparação entre suas variáveis.
  - Realizar operações lógicas entre suas variáveis.
  - Realizar operações em nível de bits com suas variáveis

Operadores aritméticos

- São aqueles que operam sobre números (valores, variáveis, constantes ou chamadas de funções) e/ou expressões e têm como resultados valores numéricos
  - Note que os operadores aritméticos são sempre usados em conjunto com o operador de atribuição.

Operador	Significado	Exemplo
+	Adição de dois valores	z = x + y
-	Subtração de dois valores	z = x – y
*	Multiplicação de dois valores	z = x * y
/	Quociente de dois valores	z = x / y
%	Resto de uma divisão	z = x % y

Operadores aritméticos

- Podemos devolver o resultado para uma outra variável ou para um outro comando ou função que espere receber um valor do mesmo tipo do resultado da operação, no caso, a função cout.

```
#include<iostream>

using namespace std;

int main(){
    int a= 3;
    int b= 5;
    cout<< a << endl << b << endl << a+b;

    return 0;

}
```

Operadores aritméticos

- IMPORTANTE
  - As operações de multiplicação, divisão e resto são executadas antes das operações de adição e subtração. Para forçar uma operação a ser executada antes das demais, ela é colocada entre parênteses
    - **z = x * y + 10;**
    - **z = x * (y + 10);**
  - O operador de subtração também pode ser utilizado para inverter o sinal de um número
    - **x = –y;**
  - Neste caso, a variável **x** receberá o valor de **y** multiplicado por **–1**, ou seja,
    - **x = (–1) * y;**

Operadores relacionais

- São aqueles que verificam a magnitude (qual é maior ou menor) e/ou igualdade entre dois valores e/ou expressões.
  - Os operadores relacionais são operadores de comparação de valores
  - Retorna ***verdadeiro*** (*1*) ou ***falso*** (*0*)

Operador	Significado	Exemplo
>	Maior do que	X > 5
>=	Maior ou igual a	X >= Y
<	Menor do que	X < 5
<=	Menor ou igual a	X <= Z
==	Igual a	X == 0
!=	Diferente de	X != Y

Operadores lógicos

- Certas situações não podem ser modeladas utilizando apenas os operadores aritméticos e/ou relacionais
  - Um exemplo bastante simples disso é saber se determinada variável x está dentro de uma faixa de valores.
  - Por exemplo, a expressão matemática
    - $0 < x < 10$
  - indica que o valor de x deve ser maior do que 0 (zero) e também menor do que 10

Operadores lógicos

- Os operadores lógicos permitem representar situações lógicas unindo duas ou mais expressões relacionais simples em uma composta
  - Retorna ***verdadeiro*** (*1*) ou ***falso*** (*0*)
- Exemplo
  - A expressão ***0 < x < 10***
  - Equivale a ***(x > 0) && (x < 10)***

Operador	Significado	Exemplo
&&	Operador **E**	(x > 0) && (x < 10)
||	Operador **OU**	(a == 'F') || (b != 32)
!	Operador **NEGAÇÃO**	!(x == 10)

Operadores lógicos

- Tabela verdade
  - Os termos ***a*** e ***b*** representam o resultado de duas expressões relacionais

a	b	!a	!b	a && b	a || b
0	**0**	**1**	**1**	**0**	**0**
0	**1**	**1**	**0**	**0**	**1**
1	**0**	**0**	**1**	**0**	**1**
1	**1**	**0**	**0**	**1**	**1**

Operadores de pré e pós-incremento/decremento

- Esses operadores podem ser utilizados sempre que for necessário somar uma unidade (incremento) ou subtrair uma unidade (decremento) a determinado valor

Operador	Significado	Exemplo	Resultado
++	incremento	++x ou x++	x = x + 1
--	decremento	--x ou x--	x = x - 1

Operadores de pré e pós-incremento/decremento

- Qual a diferença em usar antes ou depois da variável?

Operador	Significado	Resultado
++x	pré-incremento	soma +1 à variável x antes de utilizar seu valor
x++	pós-incremento	soma +1 à variável x depois de utilizar seu valor
--x	pré-decremento	subtrai −1 da variável x antes de utilizar seu valor
x--	pós-decremento	subtrai −1 da variável x depois de utilizar seu valor

tem importância se o operador for usado sozinho

- Porém, se esse operador for utilizado dentro de uma expressão aritmética, a diferença entre os dois operadores será evidente

Operadores de pré e pós-incremento/decremento

- Essa diferença de sintaxe no uso do operador não tem importância se o operador for usado sozinho
  - Porém, se utilizado dentro de uma expressão aritmética, a diferença entre os dois operadores será evidente

```
#include<iostream>

using namespace std;

int main(){

    int x, y;
    x=10;
    y= x++;
    cout << x << endl; //11
    cout << y << endl; //10
    y=++x;
    cout << x << endl; //12
    cout << y << endl; //12


    return 0;

    }
```

Exercícios

- 1 – Escreva um programa que calcule a área de um triângulo equilátero, lendo o valor do lado “a” na tela.

Operadores de atribuição simplificada

- Muitos operadores são sempre usados em conjunto com o operador de atribuição.
  - Para tornar essa tarefa mais simples, a linguagem C permite simplificar algumas expressões

Operador	Significado	Exemplo		
+=	Soma e atribui	x += y	igual a	x = x + y
-=	Subtrai e atribui	x -= y	igual a	x = x – y
*=	Multiplica e atribui	x *= y	igual a	x = x * y
/=	Divide e atribui o quociente	x /= y	igual a	x = x / y
%=	Divide e atribui o resto	x %= y	igual a	x = x % y

Operadores

- Exercício
  - Diga o resultado das variáveis x, y e z depois da seguinte sequência de operações:

```
int x, y, z;
x = y = 10;
z = ++x;
x -= x;
y++;
x = x + y - (z--);
```

Operadores

- Exercício
  - Diga o resultado das variáveis x, y e z depois da seguinte sequência de operações:

```
int x, y;
int a = 14, b = 3;
float z;
x = a / b;
y = a % b;
z = y / x;
```

Operadores

- Exercício
  - Diga se as seguintes expressões serão verdadeiras ou falsas:

```
int x = 7;
(x > 5) || (x > 10)
(!(x == 6) && (x >= 6)
```

Precedência dos Operadores

MAIOR PRECEDÊNCIA	
++ --	Pré-incremento/decremento
()	Parênteses (chamada de função)
[]	Elemento de array
.	Elemento de struct
->	Conteúdo de elemento de ponteiro para struct
++ --	Pós-incremento/decremento
+ -	Adição e subtração unária
! ~	Não lógico e complemento bit a bit
(tipo)	Conversão de tipos (*type cast*)
*	Acesso ao conteúdo de ponteiro
&	Endereço de memória do elemento
sizeof	Tamanho do elemento
* / %	Multiplicação, divisão e módulo (resto)
+ -	Adição e subtração
<< >>	Deslocamento de bits à esquerda e à direita
< <=	"Menor do que" e "menor ou igual a"
> >=	"Maior do que" e "maior ou igual a"
== !=	"Igual a" e "diferente de"
&	E bit a bit
^	OU exclusivo
|	OU bit a bit
&	E lógico
||	OU lógico
?:	Operador ternário
=	Atribuição
+= -=	Atribuição por adição ou subtração
*= /= %=	Atribuição por multiplicação, divisão ou módulo (resto)
<<= >>=	Atribuição por deslocamento de bits
&= ^= |=	Atribuição por operações lógicas
,	Operador vírgula
MENOR PRECEDÊNCIA	